

# MONGO DB

## Lezione III

# RIPASSO

# RESOCONTO ESERCIZIO

# AND THE WINNING TEAM IS

Voto basato sull'esercizio!

1 - I Mongolini - 3 pt

2 - Scobyboodidoo - 2 pt

3 - Jenkis Khan - 1 pt

Non qualificati - I Budini, I Pavesini

# Classifica

1 - I Mongolini - 5 pt

2 - Scobyboodidoo - 4 pt

3 - Jenkis Khan / I Budini - 3 pt

4 - I Pavesini - 2 pt

# AND THE WINNING CLASS IS

Voto basato sulla media degli esercizi!

E il risultato è: pareggio!

Pixel (media più bassa ma hanno consegnato tutti)

Zorin (Media più alta ma hanno consegnato 3/5)

Conducono i Pixel per 2-1

# Gli indici - Cosa Sono

Un indice in MongoDB è una struttura in cui vengono conservate porzioni di dati di una collezione.

Per gli indici vengono di solito sfruttate delle strutture dati efficienti e veloci da scansionare in modo da rendere le query estremamente performanti.

Per ogni collezione potremo creare diversi indici in base al tipo di query da eseguire.

Gli indici saranno poi usati interamente o come struttura di supporto per individuare i dati in modo più efficiente, riducendo così il numero di documenti da esaminare.

# Gli indici - Come funzionano

Prendiamo in esempio una find: db.persone.**find**{nome: "Marco"})

SENZA INDICE

Persone:  
{...}  
{...}  
{...}  
{...}  
{...}  
{...}  
{...}  
{...}

C  
O  
L  
L  
S  
C  
A  
N



Risultato:

{...}  
{...}

Scansiona TUTTI i documenti e  
poi filtra

CON INDICE nome

Lista  
ordinata  
"Marco"  
"Daniele"  
"Luca"  
"Gianni"

I  
X  
S  
C  
A  
N



Salta direttamente ai risultati filtrati

# Gli indici - Come funzionano

Come abbiamo visto MongoDB in assenza di indici esegue SEMPRE un COLLECTIONSCAN, cioè va a verificare l'intera collection, documento per documento.

Per collection molto grandi questo può rappresentare un problema di performance.

Per velocizzare il processo possiamo creare un indice (o INDEX).

L'indice non è un rimpiazzo per una collection ma una "aggiunta". Esso infatti viene creato e mantenuto parallelamente alla collection.

Sostanzialmente l'indice è una lista ORDINATA di valori relativi ai campi che indichiamo.

# Gli indici - Come funzionano

Ognuno di questi oggetti ha un “puntatore” che mette in riferimento la lista dell’indice con il relativo documento completo nella collection.

Questo permette a MongoDB di effettuare un Index Scan, utilizzando la più compatta lista e solo successivamente puntare direttamente e velocemente al documento interessato.

Gli indici sono uno strumento imprescindibile in Mongo ma non dobbiamo abusarne.

Una volta creati gli indici vengono mantenuti da MongoDB, aggiornandoli ad ogni Insert. Va da sé che questo appesantisce il processo.

Valutate sempre il contesto e non create sovrastrutture che non servono!

# Indici singoli

Vediamo ora la sintassi per creare un indice.

```
db.nomeCollection.createIndex( {nomeCampo: 1/-1 (ASC/DESC) })
```

Possiamo specificare ordine ascendente o discendente.

Per ora limitiamoci a indici con un singolo campo.

# Explain

Abbiamo visto che gli indici teoricamente dovrebbero velocizzare l'esecuzione delle nostre query, come possiamo però esserne sicuri?

L'istruzione Explain assolve a questa funzione.

Vediamo la sintassi:

```
db.nomCol.explain().find()
```

Tramite questo comando otteniamo un json di output contenente svariate informazioni relative alla query appena lanciata.

Possiamo richiedere ulteriori informazioni passando un attributo in questo modo:

```
db.nomCol.explain("executionStats").find()
```

Abbiamo quindi un modo pratico per testare le nostre query.

# Restrizioni degli indici

Abbiamo capito che gli indici velocizzano la lettura (`find`) ma appesantisce l'inserimento (`insert`)

Che sia solo questo il possibile lato negativo del creare un indice?

Quale potrebbe essere un possibile problema?

# Indici composti

Uno degli scopi che si prefiggono gli indici è quello di ridurre il numero di elementi dell'indice prima e documenti poi che andiamo ad analizzare.

Meno documenti analizziamo e più veloce sarà la nostra query!

Come possiamo però ottenere questo risultato?

Uno dei metodi che abbiamo è la creazione di indici composti, dove più chiavi sono coinvolte.

Attenzione → 1 indice con più chiavi è diverso da molti indici a singola chiave!

Ricorda → L'ordine in cui compaiono nella creazione dell'indice è importante, l'ordine in cui compaiono nella find no!

# MONGO DB

## Lezione III

### Esercizio I singolo

# Esercizio I

Utilizzando la collection fornita (Person) crea l'indice più consono da applicare tenendo presente che:

La piattaforma deve spesso, per campagna pubblicitaria, estrarre i numeri di telefono degli abitanti di singole città.

Saltuariamente gli utenti vengono estratti a fini statistici in relazione al loro territorio

Attenzione! Puoi e devi usare explain per verificare le prestazioni del tuo indice!

# Indici e sorting

Un altro utilizzo degli indici riguarda il sorting.

MongoDB per eseguire questa operazione carica in memoria tutti i documenti che ha trovato con il comando `find`.

Il problema è che MongoDB ha un limite di 32 megabytes. Su db piccoli questo non rappresenta un problema ma su DB grandi con milioni di documenti provate a immaginare cosa può accadere.

In questo gli indici vengono in nostro aiuto. Le liste che andiamo a creare con gli indici sono ordinate e contengono pochi dati.

MongoDB può quindi usarle per ordinare il risultato delle nostre `find`.

Questo a patto che il campo su cui andiamo a fare il sort sia presente nell'indice ovviamente

# Default index

Se siete stati attenti avrete notato che esiste già un indice presente nelle vostre collection.

Questo rappresenta l'id dei singoli campi.

Viene creato di default alla creazione della collection con vincolo di unicità.

Possiamo applicare questo vincolo anche ad altri campi

# Configurazione

Con il comando `createIndex` oltre ai campi interessati possiamo passare un ulteriore json di opzioni.

Ad esempio per creare un vincolo di unicità la sintassi sarà la seguente:

```
db.nomeCollection.createIndex(  
  {nomeCampo: 1/-1}, ← lista campi  
  {unique: true} ← opzioni  
)
```

Come abbiamo visto nell'esempio gli indici sono retroattivi, le loro condizioni devono essere rispettate già al momento della loro creazione.

Questo ha senso in quanto impedisce errori logici. Ricordate sempre che su db molto grandi il cherry picking dei dati è virtualmente impossibile.

# Filtri parziali

Può capitare di avere contesti in cui non serve sempre verificare tutto il contenuto di una collection e del suo indice.

Poniamo l'esempio di una azienda che faccia marketing per prodotti da barba. In questo contesto serve andare a cercare solo gli individui di sesso maschile, ordinati per età. L'indice sul campo "gender" e "age" sicuramente aiuterebbe ma la nostra query andrebbe COMUNQUE a cercare tra tutti gli elementi.

Per velocizzare ulteriormente possiamo usare un'altra opzione nel comando `createIndex`:

```
db.nomeCollection.createIndex(  
  {  
    age: 1  
  },  
  {  
    partialFilterExpression:  
      {  
        gender: "male"  
      }  
  }  
)
```

# Indici Time to live (TTL)

Esistono anche contesti in cui non è necessario mantenere i dati in un DB per tempi prolungati. File di log, dati di sessione, sono esempi di dati che una volta utilizzati possono essere eliminati.

MongoDB permette di rendere questa operazione automatica tramite l'opzione `expireAfterSeconds`.

Attenzione: va usato solo su campi data!

```
db.nomeCollection.createIndex(  
  {createdAt: 1}, ← campo data  
  {expireAfterSeconds: 1}) ← numero secondi
```

Una volta trascorso il numero di secondi indicati MongoDB automaticamente elimina il documento.

Attenzione: questo è uno dei pochi casi in cui esiste una valenza retroattiva!

# Covered Query

Per creare query efficienti abbiamo visto che dobbiamo ridurre il numero di elementi dell'indice e di documenti coinvolti.

Un modo per riassumere sommariamente una buona costruzione di indici è il seguente →

Numero di chiavi di indice esaminate

Più vicino possibile al numero di

numero di documenti pari a zero



Numero di documenti esaminati

Più vicino possibile al numero di

numero di documenti pari a zero



Numero di documenti restituiti

# Covered Query

Come è possibile che il numero di documenti esaminati sia pari a zero?

È nei documenti che vengono salvati i dati, senza quelli come potremmo accedere ai dati?

La risposta è nella clausola di projection!

Questa ci permette di filtrare il risultato indicando solo determinati campi.

Se indichiamo solo i campi presenti nell'indice Mongo non avrà bisogno di andare a interrogare la collection in quanto ha già tutto quello di cui ha bisogno.

# Rejected plan

Come fa MongoDB a decidere quale piano di esecuzione utilizzare?

L'approccio di Mongo è tanto banale quanto efficace

Si basa su un semplice trial and error. Il db individua tutti i possibili piani e li lancia su un piccolo "campione" (100 documenti).

Questo però ci porta ad un altro problema. Se eseguiamo questa operazione continuamente non andiamo ad appesantire l'esecuzione?

Mongo ha risolto tenendo in cache questo "test". Il risultato viene tenuto in memoria e se la stessa query (con stessa intendo IDENTICA) viene lanciata Mongo sa già quale piano usare.

Questa cache non rimane in eterno, esistono delle condizioni che la nuova esecuzione del test.

# Rejected plan

Limite di scrittura → 1000 documenti

L'indice viene ricreato

Altri indici vengono aggiunti o rimossi

Il server di Mongo viene riavviato

# Best Practice

Saltuariamente o a cadenza regolare, all'aumentare della base dati o a seguito di modifiche sostanziali del db e del suo utilizzo le prestazioni delle query andrebbero controllate manualmente!

Un sacco di elementi possono influire sulle performance e il DB ha i suoi limiti, l'intervento umano (al momento) è indispensabile per mantenere il DB performante!

# MONGO DB

## Lezione III

### Esercizio Il singolo

# Esercizio II

- 1: Crea una collection (e relativa query) che rappresenti i prodotti di uno shop online di album musicali (ogni prodotto può essere digital download o vinile ma non entrambi) e crea un indice che produca una covered query.
  - 2: Crea una collection (e relativa query) della sessione di navigazione dell'utente e un indice TTL di 10 minuti per documento.
  - 3 :Crea una collection (e relativa query) che rappresenti il carrello di questo shop e crea l'indice che svuoti il carrello dopo un giorno dall'ultima interazione dell'utente.
  - 4: Usando la collection prodotti crea l'indice più performante considerando che la maggior parte dei clienti preferisce l'acquisto analogico
- NOTA BENE: Per far funzionare correttamente i punti 2 e 3 serve una particolare attenzione nella query!

# Indici multi chiave

È possibile creare indici su campi di tipo array. Questi vengono definiti indici multi chiave (multi-key indexes).

Ad ogni elemento dell'array corrisponderà un singolo elemento dell'indice. Questo implica che gli indici multi chiave sono potenzialmente molto più grandi rispetto a quelli standard.

Questo va tenuto bene in considerazione

Attenzione alla differenza tra indicare come indice un array come campo e un oggetto contenente un array!

Mongo necessita una precisa indicazione di quali campi intendete usare, per sapere come stoccare questi dati nell'indice.

# Indici di testo

Questi indici vengono usati per stoccare le parole contenute in uno o più campi.

L'indice di testo (text indexes) è di fatto un indice multichiave ma con delle sue regole.

Il testo (in base alla lingua) viene spezzettato in singole parole che vengono salvate nel relativo oggetto.

Prendiamo come esempio questa stringa:

"Francesco is the best teacher you could ask for"

Non tutte le parole vengono salvate nell'indice ma solo quelle che vengono considerate parole "chiave".

Le parole che vengono salvate saranno una cosa del tipo : Francesco - best - teacher - could - ask

In questa maniera si escludono parole "comuni" e che falserebbero la ricerca.

# Indici di testo

Vediamo un po di sintassi.

```
db.nomeCollection.createIndex(  
{nomeCampo: "text"}) ← Keyword
```

Come vedete al posto del `1/-1` viene passata una keyword “text”. Questo serve a mongo per l’operazione discussa nella slide precedente.

Se passiamo `1/-1` un indice verrà creato ma con chiave l’intero testo.

Ora possiamo creare una query di ricerca.

```
db.nomeCollection.find(  
{$text:  
 {  
   $search : "testo da cercare"  
 }  
})
```

Come avrete notato non specifichiamo il campo in cui fare questa ricerca, come mai?

# Indici di testo

Una collection può avere un solo indice di testo.

Questo non significa che possiamo inserire un solo campo, possiamo infatti passare più campi all'indice.

Il testo da cercare è una stringa. In caso di più parole separate da spazi queste verranno cercate singolarmente.

Nel testo da cercare se in una parola indichiamo il - questa esclude il risultato, il "" trova corrispondenza esatta. Ricorda qualcosa?

```
db.nomeCollection.find(  
{$text:  
{  
$search : "-testo da \\"cercare\\\""  
}  
})
```

# Indici di testo, punteggi e sorting

Questo ci permette di effettuare ricerche testuali.

Possiamo andare più a fondo e verificare il punteggio di ricerca. Ogni ricerca avrà infatti un valore numerico che rappresenta quanto il campo sia vicino alle parole cercate.

Con la query qui indicata potete verificare il “risultato” di questo punteggio e ordinare secondo questo valore.

```
db.nomeCollection.find(  
{$text:  
{  
$search : "testo da cercare"  
},  
{  
score: {$meta: "textScore"}  
}  
})
```

# Combinare indici di testo

Come detto inizialmente può esistere un unico indice di testo per ogni collection.

Questo può avere più campi

```
db.nomeCollection.createIndex(  
  {nomeCampo1: "text",  
   nomeCampo2: "text",  
   ...})
```

Possiamo infine decidere arbitrariamente che ogni campo abbia più “peso” nella ricerca.

Per esempio potremmo volere che il nome di un prodotto valga più della sua descrizione.

# Indici di testo e peso

Per assegnare un “peso” (weight) ad ogni campo la sintassi è la seguente:

```
db.nomeCollection.createIndex(  
  {campo1: "text",  
   campo2: "text",  
   ...},  
  {weights : {campo1: 1, campo2: 10}}  
 )
```

Questo torna particolarmente utile in casi di filtri di ricerca particolarmente approfonditi.

In questo modo il campo2 avrà una valenza maggiore in caso di corrispondenza.

# Utilizzo in background

Di default la creazione di indici può essere un'operazione molto lunga in caso di DB di grandi dimensioni.

Questo pone la collection in uno stato di "lock", che ne impedisce l'utilizzo fino ad operazione conclusa.

Va da sé che bisogna porre particolare attenzione alle operazioni che si eseguono sui DB di produzione.

Possiamo indicare a MongoDB di eseguire l'operazione in background.

```
db.nomeCollection.createIndex(  
  {campo1: "text"},  
  {background : true} ← opzioni  
)
```

Come mai non utilizziamo sempre questa funzione?

# MONGO DB

## Lezione III

### Esercizio III di gruppo

# Esercizio III

Rivedi l'esercizio precedente con le note e gli errori visti durante la correzione dell'esercizio.

Crea gli indici adatti a questo DB, almeno:

- 1 indice semplice
- 1 indice composto
- 1 indice multichiave
- 1 indice di testo
- 1 indice parziale
- 1 esempio di covered query.

Queste non devono essere a caso. Per ogni indice spiega perchè lo hai fatto e il risultato che ti aspetti.